**A Deep Dive on Clustered Index**

[Brahmanand Shukla](https://www.sqlservercentral.com/author/brahmanand-shukla), 2020-05-07 (first published: 2020-04-27)

FROM: <https://www.sqlservercentral.com/blogs/a-deep-dive-on-clustered-index>

Background

This article will follow a practical approach to choose the right candidate for clustered index. It’ll investigate through the best practices recommended for clustered index, with the help of an experiment.

Primary Key and Clustered Index

Primary Key and Clustered Index are different from each other. Clustered Index should not be confused with Primary Key. Primary Key can be created with / without Clustered / Nonclustered Index.

Description of terminologies used

**Natural Clustered Key:** Column or set of columns forming the clustered index, naturally part of table having some business meaning associated with it.

**Surrogate Clustered Key:** Artificial column such as Identity or uniqueidentifier column, forming the clustered index. We’ve used Identity column as Surrogate Clustered Key in our experiment.

Clustered Index best practices

1. It should be as **narrow**as possible in terms of the number of bytes it stores.
2. It should be **unique** to avoid SQL Server to add Uniquefier to duplicate key values.
3. It should be **static**, ideally, never updated like Identity column.
4. It should be **ever-increasing**, like Identity column to avoid fragmentation and improve the write performance.

**Why should we always have a Clustered Index?**

1. Clustered Index out-performs NonClustered Index, since it stores the data for every row. Data page is stored in the leaf node of the Clustered Index.
2. Data is sorted based on clustered index key values and then stored.
3. The only time the data is stored in sorted order when the table contains a clustered index. Sort operation is most costly operator in SQL Server.
4. Since data is stored in sorted order, hence sorting on the clustered key column avoids the sort operator, and makes it best choice for sorting.
5. It helps the Database engine to lock a particular row, instead of a Page or a Table. Row level locking can be only achieved if table has Clustered Index. SQL Server applies lock on Row, Page or Table to ensure consistency.

What should be our Clustered Index?

By looking at the best practices, it’s very hard to find a Natural Key that can satisfy all the four recommendations.

Prima facie, it seems Surrogate Key, such as an Identity column seems to be the only best candidate for Primary Key.

Wait, if Surrogate Key is the best candidate then how to deal with following overheads?

**How to minimize Sorting overhead?**

Clustered Index column should be referred in Sort operation in the same sort order as defined in the index. Surrogate Key for e.g. Identity column is hardly the sorting requirement.

**How to minimize Lock escalation – Page or Table lock?**

All the DELETE and UPDATE should happen on Clustered Index Key column. There is hardly a requirement to delete / update based on Surrogate Key.

Alternate approach could be – take all the clustered key column values in a temp table and then using join or sub-query use it in delete / update statement. This is an additional overhead.

**How to avoid Key Lookup?**

Create nonclustered [covering index](https://docs.microsoft.com/en-us/sql/relational-databases/indexes/create-indexes-with-included-columns?view=sql-server-ver15). This is another overhead.

What we’ll achieve through this experiment?

We’ll compare the following parameters. It’ll be then left to us to decide what best suits to us, depending upon the requirement and workload.

* Execution Time
  + Avg. Read Execution Time
  + Avg. Write Execution Time
* Index size and read/write count
  + Index size (KB)
  + num\_of\_writes
  + num\_of\_reads
* Lock/Latch count/wait count/wait time
  + row\_lock\_count
  + page\_lock\_count
  + page\_lock\_wait\_count
  + page\_lock\_wait\_in\_ms
  + index\_lock\_promotion\_attempt\_count
  + page\_latch\_wait\_count
  + page\_latch\_wait\_in\_ms
  + tree\_page\_latch\_wait\_count
  + tree\_page\_latch\_wait\_in\_ms
* Index usage stats
  + user\_seeks
  + user\_scans
  + user\_updates

Workload considered for this test

IoT (Internet of Things) and IIot (Industrial Internet of Things) domains are leading domains. These domains deals with huge volume of data. Data volume is huge in both read as well write, since it deals with machines.

A machine has multiple components and every components has multiple sensors and every sensor sends multiple messages per seconds/milliseconds.

Timestamp is the key attribute for the calculation of the KPI’s to show the real-time analytics.

We’ll have a very simple example with one table, that holds the Coffee Vending Machine’s Sensor Data. Every second, 1000’s of records of multiple machines gets written into this table. The scenario here pertains to an application which gets refreshed every 10 seconds and reads past few minutes of data to render analytics over the UI.

Following aspects are considered in the workload:

* Committed Read to reproduce the locking
* Uncommitted Read / Dirty Read
* Insert new rows
* Update existing rows to simulate the Lock Escalation
* Concurrency with the help of 2 jobs each for both read and write

Script of table schema

CREATE TABLE Coffee\_Vending\_Machine\_Data

(

ID NUMERIC IDENTITY(1,1) NOT NULL

, Date\_Time DATETIME

, MachineID INT

, Sensor1\_Value INT

, Sensor2\_Value INT

, Sensor3\_Value INT

, Sensor4\_Value INT

)

Script to load table with lot of data

SET NOCOUNT ON

DECLARE @RowID INT = 1

, @MachineID INT

, @Date\_Time DATETIME

, @Sensor1\_Value INT

, @Sensor2\_Value INT

, @Sensor3\_Value INT

, @Sensor4\_Value INT

WHILE (@RowID <= 1773885)

BEGIN

SET @Date\_Time = GETDATE();

SET @MachineID = ROUND((RAND() \* 10), 0)

SET @Sensor1\_Value = ROUND((RAND() \* 100), 0)

SET @Sensor2\_Value = ROUND((RAND() \* 100), 0)

SET @Sensor3\_Value = ROUND((RAND() \* 100), 0)

SET @Sensor4\_Value = ROUND((RAND() \* 100), 0)

INSERT INTO Coffee\_Vending\_Machine\_Data (Date\_Time, MachineID, Sensor1\_Value, Sensor2\_Value, Sensor3\_Value, Sensor4\_Value)

VALUES (@Date\_Time, @MachineID, @Sensor1\_Value, @Sensor2\_Value, @Sensor3\_Value, @Sensor4\_Value)

SET @RowID = @RowID + 1;

END

Other prerequisites

Script of table to log the execution time

CREATE TABLE log\_simulation\_job\_execution

(

LogID INT NOT NULL IDENTITY(1,1)

, JobName VARCHAR(100)

, Start\_DateTime DATETIME

, End\_DateTime DATETIME

, Duration\_MS AS DATEDIFF(MS, Start\_DateTime, End\_DateTime)

)

Script of Read Stored Procedure

CREATE PROCEDURE [dbo].[usp\_read\_sensor\_data]

(

@Start\_Date\_Time DATETIME

)

AS

BEGIN

SET NOCOUNT ON;

DECLARE @Job\_Start\_DateTime DATETIME

, @Job\_End\_DateTime DATETIME;

SET @Job\_Start\_DateTime = GETDATE();

SELECT AVG(Sensor1\_Value)

, MIN(Sensor2\_Value)

, MAX(Sensor3\_Value)

, AVG(Sensor4\_Value)

, COUNT(1)

FROM Coffee\_Vending\_Machine\_Data (NOLOCK)

WHERE MachineID = FLOOR(RAND() \* 10)

AND Date\_Time BETWEEN @Start\_Date\_Time AND DATEADD(MINUTE, FLOOR(RAND() \* 10), @Start\_Date\_Time);

SET @Job\_End\_DateTime = GETDATE();

INSERT INTO log\_simulation\_job\_execution

(

JobName

, Start\_DateTime

, End\_DateTime

)

VALUES

(

'Simulate\_Read\_Load'

, @Job\_Start\_DateTime

, @Job\_End\_DateTime

);

END

Script of Write Stored Procedure

CREATE PROCEDURE [dbo].[usp\_write\_sensor\_data]

AS

BEGIN

SET NOCOUNT ON;

DECLARE @Job\_Start\_DateTime DATETIME

, @Job\_End\_DateTime DATETIME;

SET @Job\_Start\_DateTime = GETDATE();

DECLARE @RowID INT = 1

, @MachineID INT

, @Date\_Time DATETIME

, @Sensor1\_Value INT

, @Sensor2\_Value INT

, @Sensor3\_Value INT

, @Sensor4\_Value INT;

DECLARE @Sensor1\_Value\_Avg INT

, @Sensor2\_Value\_Avg INT

, @Sensor3\_Value\_Avg INT

, @Sensor4\_Value\_Avg INT

, @Machine\_ID\_Min INT

, @Machine\_ID\_Max INT;

SELECT @Sensor1\_Value\_Avg = AVG(Sensor1\_Value)

, @Sensor2\_Value\_Avg = AVG(Sensor2\_Value)

, @Sensor3\_Value\_Avg = AVG(Sensor3\_Value)

, @Sensor4\_Value\_Avg = AVG(Sensor4\_Value)

, @Machine\_ID\_Min = MIN(MachineID)

, @Machine\_ID\_Max = MAX(MachineID)

FROM Coffee\_Vending\_Machine\_Data;

WHILE (@RowID <= 1000)

BEGIN

SET @Date\_Time = GETDATE();

SET @MachineID = ROUND((RAND() \* 10), 0)

SET @Sensor1\_Value = ROUND((RAND() \* 100), 0)

SET @Sensor2\_Value = ROUND((RAND() \* 100), 0)

SET @Sensor3\_Value = ROUND((RAND() \* 100), 0)

SET @Sensor4\_Value = ROUND((RAND() \* 100), 0)

INSERT INTO Coffee\_Vending\_Machine\_Data (Date\_Time, MachineID, Sensor1\_Value, Sensor2\_Value, Sensor3\_Value, Sensor4\_Value)

VALUES (@Date\_Time, @MachineID, @Sensor1\_Value, @Sensor2\_Value, @Sensor3\_Value, @Sensor4\_Value)

SET @RowID = @RowID + 1;

END

-- This UPDATE statement may not make sense, but still keeping it here, to properly simulate the READ/WRITE overhead.

UPDATE Coffee\_Vending\_Machine\_Data

SET Sensor1\_Value = Sensor1\_Value + @Sensor1\_Value\_Avg

, Sensor2\_Value = Sensor2\_Value + @Sensor2\_Value\_Avg

, Sensor3\_Value = Sensor3\_Value + @Sensor3\_Value\_Avg

, Sensor4\_Value = Sensor4\_Value + @Sensor4\_Value\_Avg

WHERE MachineID BETWEEN @Machine\_ID\_Min AND @Machine\_ID\_Max

AND @Date\_Time BETWEEN @Job\_Start\_DateTime AND GETDATE();

SET @Job\_End\_DateTime = GETDATE();

INSERT INTO log\_simulation\_job\_execution

(

JobName

, Start\_DateTime

, End\_DateTime

)

VALUES

(

'Simulate\_Write\_Load'

, @Job\_Start\_DateTime

, @Job\_End\_DateTime

);

END

Script of Read Job 1

USE [msdb]

GO

/\*\*\*\*\*\* Object: Job [Simulate\_Read\_Load] \*\*\*\*\*\*/

BEGIN TRANSACTION

DECLARE @ReturnCode INT

SELECT @ReturnCode = 0

/\*\*\*\*\*\* Object: JobCategory [[Uncategorized (Local)]] \*\*\*\*\*\*/

IF NOT EXISTS (SELECT name FROM msdb.dbo.syscategories WHERE name=N'[Uncategorized (Local)]' AND category\_class=1)

BEGIN

EXEC @ReturnCode = msdb.dbo.sp\_add\_category @class=N'JOB', @type=N'LOCAL', @name=N'[Uncategorized (Local)]'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

END

DECLARE @jobId BINARY(16)

EXEC @ReturnCode = msdb.dbo.sp\_add\_job @job\_name=N'Simulate\_Read\_Load',

@enabled=0,

@notify\_level\_eventlog=0,

@notify\_level\_email=0,

@notify\_level\_netsend=0,

@notify\_level\_page=0,

@delete\_level=0,

@description=N'No description available.',

@category\_name=N'[Uncategorized (Local)]',

@owner\_login\_name=<your owner login name>, @job\_id = @jobId OUTPUT

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

/\*\*\*\*\*\* Object: Step [1] \*\*\*\*\*\*/

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobstep @job\_id=@jobId, @step\_name=N'1',

@step\_id=1,

@cmdexec\_success\_code=0,

@on\_success\_action=1,

@on\_success\_step\_id=0,

@on\_fail\_action=2,

@on\_fail\_step\_id=0,

@retry\_attempts=0,

@retry\_interval=0,

@os\_run\_priority=0, @subsystem=N'TSQL',

@command=N'DECLARE @Date\_Time DATETIME

SELECT @Date\_Time = MIN(Date\_Time)

FROM Coffee\_Vending\_Machine\_Data

EXEC [dbo].[usp\_read\_sensor\_data] @Start\_Date\_Time = @Date\_Time;

',

@database\_name=N'MyDB',

@flags=0

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_update\_job @job\_id = @jobId, @start\_step\_id = 1

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobschedule @job\_id=@jobId, @name=N'Every 10 second',

@enabled=1,

@freq\_type=4,

@freq\_interval=1,

@freq\_subday\_type=2,

@freq\_subday\_interval=10,

@freq\_relative\_interval=0,

@freq\_recurrence\_factor=0,

@active\_start\_date=20200412,

@active\_end\_date=99991231,

@active\_start\_time=0,

@active\_end\_time=235959,

@schedule\_uid=N'e041a22f-3c9f-4009-8974-5496f4536bf7'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobserver @job\_id = @jobId, @server\_name = N'(local)'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

COMMIT TRANSACTION

GOTO EndSave

QuitWithRollback:

IF (@@TRANCOUNT > 0) ROLLBACK TRANSACTION

EndSave:

GO

Script of Read Job 2

USE [msdb]

GO

/\*\*\*\*\*\* Object: Job [Simulate\_Read\_Load\_2] \*\*\*\*\*\*/

BEGIN TRANSACTION

DECLARE @ReturnCode INT

SELECT @ReturnCode = 0

/\*\*\*\*\*\* Object: JobCategory [[Uncategorized (Local)]] Script Date: 4/25/2020 5:58:59 PM \*\*\*\*\*\*/

IF NOT EXISTS (SELECT name FROM msdb.dbo.syscategories WHERE name=N'[Uncategorized (Local)]' AND category\_class=1)

BEGIN

EXEC @ReturnCode = msdb.dbo.sp\_add\_category @class=N'JOB', @type=N'LOCAL', @name=N'[Uncategorized (Local)]'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

END

DECLARE @jobId BINARY(16)

EXEC @ReturnCode = msdb.dbo.sp\_add\_job @job\_name=N'Simulate\_Read\_Load\_2',

@enabled=0,

@notify\_level\_eventlog=0,

@notify\_level\_email=0,

@notify\_level\_netsend=0,

@notify\_level\_page=0,

@delete\_level=0,

@description=N'No description available.',

@category\_name=N'[Uncategorized (Local)]',

@owner\_login\_name=<your owner login name>, @job\_id = @jobId OUTPUT

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

/\*\*\*\*\*\* Object: Step [1] \*\*\*\*\*\*/

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobstep @job\_id=@jobId, @step\_name=N'1',

@step\_id=1,

@cmdexec\_success\_code=0,

@on\_success\_action=1,

@on\_success\_step\_id=0,

@on\_fail\_action=2,

@on\_fail\_step\_id=0,

@retry\_attempts=0,

@retry\_interval=0,

@os\_run\_priority=0, @subsystem=N'TSQL',

@command=N'DECLARE @Date\_Time DATETIME

SELECT @Date\_Time = MIN(Date\_Time)

FROM Coffee\_Vending\_Machine\_Data

EXEC [dbo].[usp\_read\_sensor\_data] @Start\_Date\_Time = @Date\_Time;

',

@database\_name=N'MyDB',

@flags=0

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_update\_job @job\_id = @jobId, @start\_step\_id = 1

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobschedule @job\_id=@jobId, @name=N'Every 10 second',

@enabled=1,

@freq\_type=4,

@freq\_interval=1,

@freq\_subday\_type=2,

@freq\_subday\_interval=10,

@freq\_relative\_interval=0,

@freq\_recurrence\_factor=0,

@active\_start\_date=20200412,

@active\_end\_date=99991231,

@active\_start\_time=0,

@active\_end\_time=235959,

@schedule\_uid=N'e041a22f-3c9f-4009-8974-5496f4536bf7'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobserver @job\_id = @jobId, @server\_name = N'(local)'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

COMMIT TRANSACTION

GOTO EndSave

QuitWithRollback:

IF (@@TRANCOUNT > 0) ROLLBACK TRANSACTION

EndSave:

GO

Script of Write Job 1

USE [msdb]

GO

/\*\*\*\*\*\* Object: Job [Simulate\_Write\_Load] \*\*\*\*\*\*/

BEGIN TRANSACTION

DECLARE @ReturnCode INT

SELECT @ReturnCode = 0

/\*\*\*\*\*\* Object: JobCategory [[Uncategorized (Local)]] \*\*\*\*\*\*/

IF NOT EXISTS (SELECT name FROM msdb.dbo.syscategories WHERE name=N'[Uncategorized (Local)]' AND category\_class=1)

BEGIN

EXEC @ReturnCode = msdb.dbo.sp\_add\_category @class=N'JOB', @type=N'LOCAL', @name=N'[Uncategorized (Local)]'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

END

DECLARE @jobId BINARY(16)

EXEC @ReturnCode = msdb.dbo.sp\_add\_job @job\_name=N'Simulate\_Write\_Load',

@enabled=0,

@notify\_level\_eventlog=0,

@notify\_level\_email=0,

@notify\_level\_netsend=0,

@notify\_level\_page=0,

@delete\_level=0,

@description=N'No description available.',

@category\_name=N'[Uncategorized (Local)]',

@owner\_login\_name=<your owner login name>, @job\_id = @jobId OUTPUT

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

/\*\*\*\*\*\* Object: Step [1] \*\*\*\*\*\*/

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobstep @job\_id=@jobId, @step\_name=N'1',

@step\_id=1,

@cmdexec\_success\_code=0,

@on\_success\_action=1,

@on\_success\_step\_id=0,

@on\_fail\_action=2,

@on\_fail\_step\_id=0,

@retry\_attempts=0,

@retry\_interval=0,

@os\_run\_priority=0, @subsystem=N'TSQL',

@command=N'EXEC [dbo].[usp\_write\_sensor\_data]',

@database\_name=N'MyDB',

@flags=0

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_update\_job @job\_id = @jobId, @start\_step\_id = 1

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobschedule @job\_id=@jobId, @name=N'Every 10 seconds',

@enabled=1,

@freq\_type=4,

@freq\_interval=1,

@freq\_subday\_type=2,

@freq\_subday\_interval=10,

@freq\_relative\_interval=0,

@freq\_recurrence\_factor=0,

@active\_start\_date=20200412,

@active\_end\_date=99991231,

@active\_start\_time=0,

@active\_end\_time=235959,

@schedule\_uid=N'9a6a8f7f-f2b6-4fb5-8e2d-4058b1cb71b5'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobserver @job\_id = @jobId, @server\_name = N'(local)'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

COMMIT TRANSACTION

GOTO EndSave

QuitWithRollback:

IF (@@TRANCOUNT > 0) ROLLBACK TRANSACTION

EndSave:

GO

Script of Write Job 2

USE [msdb]

GO

/\*\*\*\*\*\* Object: Job [Simulate\_Write\_Load\_2] Script Date: 4/25/2020 5:58:27 PM \*\*\*\*\*\*/

BEGIN TRANSACTION

DECLARE @ReturnCode INT

SELECT @ReturnCode = 0

/\*\*\*\*\*\* Object: JobCategory [[Uncategorized (Local)]] \*\*\*\*\*\*/

IF NOT EXISTS (SELECT name FROM msdb.dbo.syscategories WHERE name=N'[Uncategorized (Local)]' AND category\_class=1)

BEGIN

EXEC @ReturnCode = msdb.dbo.sp\_add\_category @class=N'JOB', @type=N'LOCAL', @name=N'[Uncategorized (Local)]'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

END

DECLARE @jobId BINARY(16)

EXEC @ReturnCode = msdb.dbo.sp\_add\_job @job\_name=N'Simulate\_Write\_Load\_2',

@enabled=0,

@notify\_level\_eventlog=0,

@notify\_level\_email=0,

@notify\_level\_netsend=0,

@notify\_level\_page=0,

@delete\_level=0,

@description=N'No description available.',

@category\_name=N'[Uncategorized (Local)]',

@owner\_login\_name=<your owner login name>, @job\_id = @jobId OUTPUT

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

/\*\*\*\*\*\* Object: Step [1] \*\*\*\*\*\*/

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobstep @job\_id=@jobId, @step\_name=N'1',

@step\_id=1,

@cmdexec\_success\_code=0,

@on\_success\_action=1,

@on\_success\_step\_id=0,

@on\_fail\_action=2,

@on\_fail\_step\_id=0,

@retry\_attempts=0,

@retry\_interval=0,

@os\_run\_priority=0, @subsystem=N'TSQL',

@command=N'EXEC [dbo].[usp\_write\_sensor\_data]',

@database\_name=N'MyDB',

@flags=0

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_update\_job @job\_id = @jobId, @start\_step\_id = 1

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobschedule @job\_id=@jobId, @name=N'Every 10 seconds',

@enabled=1,

@freq\_type=4,

@freq\_interval=1,

@freq\_subday\_type=2,

@freq\_subday\_interval=10,

@freq\_relative\_interval=0,

@freq\_recurrence\_factor=0,

@active\_start\_date=20200412,

@active\_end\_date=99991231,

@active\_start\_time=0,

@active\_end\_time=235959,

@schedule\_uid=N'9a6a8f7f-f2b6-4fb5-8e2d-4058b1cb71b5'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

EXEC @ReturnCode = msdb.dbo.sp\_add\_jobserver @job\_id = @jobId, @server\_name = N'(local)'

IF (@@ERROR <> 0 OR @ReturnCode <> 0) GOTO QuitWithRollback

COMMIT TRANSACTION

GOTO EndSave

QuitWithRollback:

IF (@@TRANCOUNT > 0) ROLLBACK TRANSACTION

EndSave:

GO

Stats script 1 – Index size / avg. row size

SELECT tn.[name] AS [Table name], ix.[name] AS [Index name]

, SUM(sz.[used\_page\_count]) \* 8 AS [Index size (KB)]

, SUM(sz.row\_count) AS [Total Rows]

, CAST((SUM(sz.[used\_page\_count]) \* 8.00) / SUM(sz.row\_count) AS NUMERIC(10,4)) AS [Avg Row Size (KB)]

FROM sys.dm\_db\_partition\_stats AS sz

INNER JOIN sys.indexes AS ix

ON sz.[object\_id] = ix.[object\_id]

AND sz.[index\_id] = ix.[index\_id]

INNER JOIN sys.tables tn

ON tn.OBJECT\_ID = ix.object\_id

WHERE tn.[name] = 'Coffee\_Vending\_Machine\_Data'

GROUP BY tn.[name], ix.[name]

ORDER BY tn.[name]

Stats script 2 – Index operational and usage stats

SELECT ix.[name] AS [Index name], stat.row\_lock\_count, stat.row\_lock\_wait\_count

, stat.row\_lock\_wait\_in\_ms, stat.page\_lock\_count, stat.page\_lock\_wait\_count, stat.page\_lock\_wait\_in\_ms

, stat.index\_lock\_promotion\_attempt\_count, stat.index\_lock\_promotion\_count, stat.page\_latch\_wait\_count

, stat.page\_latch\_wait\_in\_ms, stat.page\_io\_latch\_wait\_count, stat.page\_io\_latch\_wait\_in\_ms

, stat.tree\_page\_latch\_wait\_count, stat.tree\_page\_latch\_wait\_in\_ms, stat.tree\_page\_io\_latch\_wait\_count

, stat.tree\_page\_io\_latch\_wait\_in\_ms

, ustat.user\_seeks, ustat.user\_scans, ustat.user\_lookups, ustat.user\_updates

FROM sys.dm\_db\_index\_operational\_stats(DB\_ID(),OBJECT\_ID('Coffee\_Vending\_Machine\_Data'),-1,0) stat

LEFT JOIN sys.dm\_db\_index\_usage\_stats ustat

ON ustat.object\_id = stat.object\_id

AND ustat.index\_id = stat.index\_id

INNER JOIN sys.indexes AS ix

ON stat.[object\_id] = ix.[object\_id]

AND stat.[index\_id] = ix.[index\_id]

INNER JOIN sys.tables tn

ON tn.OBJECT\_ID = ix.object\_id

Stats script 3 – Virtual file stats

SELECT files.name, stat.num\_of\_writes, stat.num\_of\_bytes\_written, stat.num\_of\_reads, stat.num\_of\_bytes\_read

FROM sys.dm\_io\_virtual\_file\_stats(DB\_ID(),NULL) stat

INNER JOIN sys.sysfiles files

ON files.fileid = stat.file\_id

Stats script 4 – Avg. execution time

SELECT JobName, COUNT(1) AS [No of execution], AVG(Duration\_MS) AS [Avg Duration (MS)]

FROM log\_simulation\_job\_execution (NOLOCK)

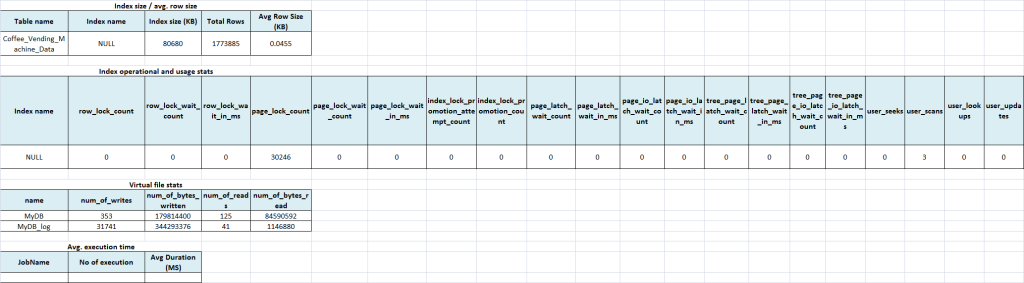
GROUP BY JobName

**Server Configuration** used for this test

This test has been performed on the server with following configurations.

* **OS :** Windows Server 2019 Datacenter 10.0 <X64> (Build 17763: ) (Hypervisor)
* **SQL Server :** SQL Server 2019 Developer Edition (64-bit)
* **RAM :** 4 GB
* **Cores :** 2 Virtual Cores
* **Disk Performance (Estimated)**
  + IOPS limit : 500
  + Throughput limit (MB/s) : 60
* **On-Premises/Cloud :** Azure Cloud

Pre-test stats

[](https://brahmanandtech.files.wordpress.com/2020/04/image-35.png)

Note: Keep all the jobs Disabled.

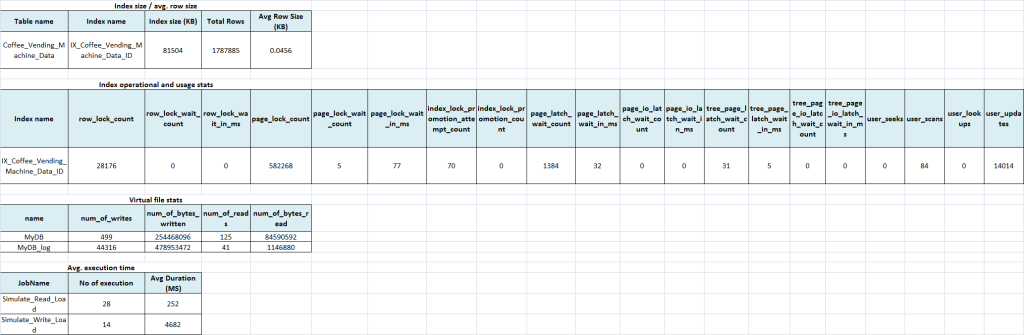
Execution#1 with Surrogate Clustered Key (without nonclustered index)

Create the index using below query.

CREATE CLUSTERED INDEX IX\_Coffee\_Vending\_Machine\_Data\_ID

ON Coffee\_Vending\_Machine\_Data (ID)

Enable all the jobs and disable it after 2 minutes. Once all the jobs are disabled, run the stats select queries.

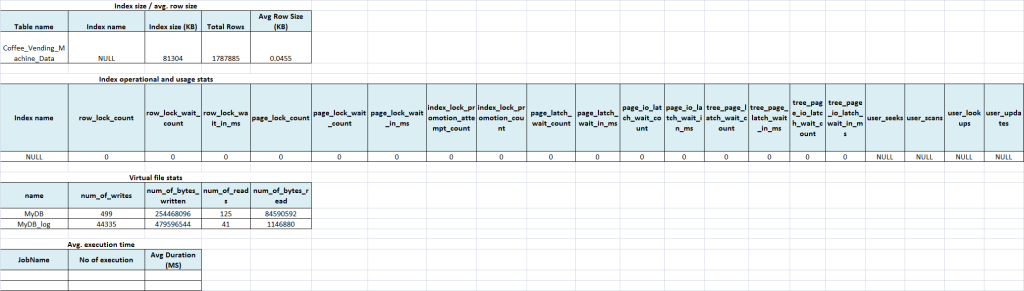
[](https://brahmanandtech.files.wordpress.com/2020/04/image-36.png)

Execution#2 with Surrogate Clustered Key (with nonclustered non covering index)

Drop the existing index using below query and run the stats select queries.

DROP INDEX IX\_Coffee\_Vending\_Machine\_Data\_ID ON Coffee\_Vending\_Machine\_Data

TRUNCATE TABLE log\_simulation\_job\_execution

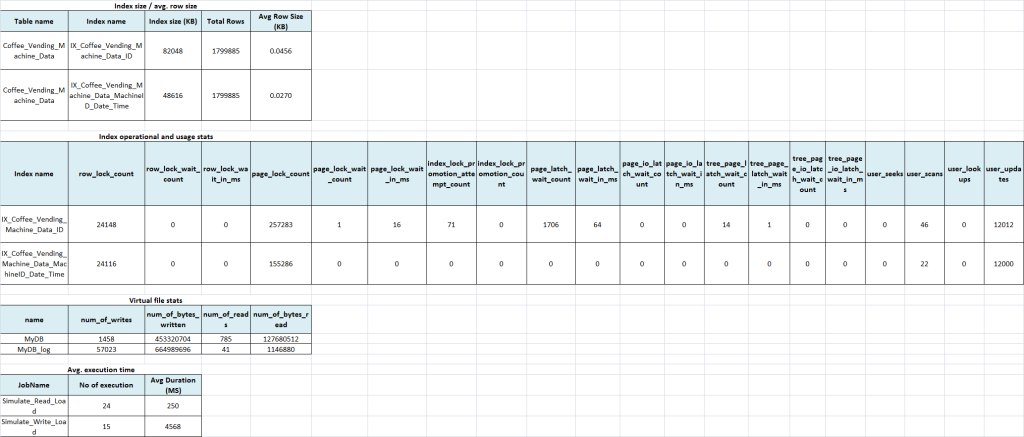
[](https://brahmanandtech.files.wordpress.com/2020/04/image-37.png)

Create the below indexes, enable all the jobs and disable it after 2 minutes. Once all the jobs are disabled run the stats select queries.

CREATE CLUSTERED INDEX IX\_Coffee\_Vending\_Machine\_Data\_ID ON Coffee\_Vending\_Machine\_Data (ID)

CREATE NONCLUSTERED INDEX IX\_Coffee\_Vending\_Machine\_Data\_MachineID\_Date\_Time

ON Coffee\_Vending\_Machine\_Data (MachineID, Date\_Time)

[](https://brahmanandtech.files.wordpress.com/2020/04/image-38.png)

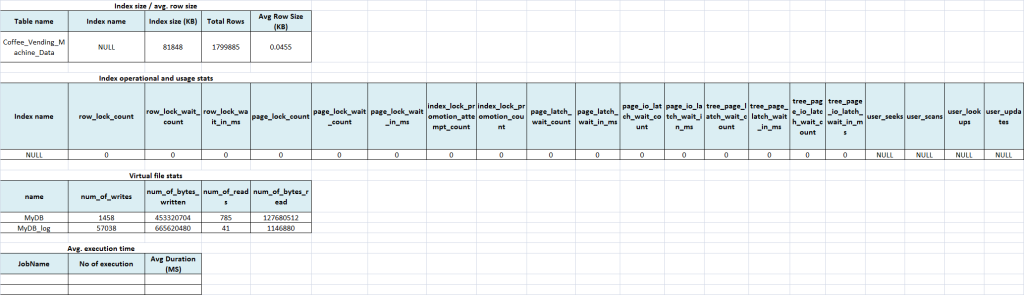
Execution#3 with Surrogate Clustered Key (with nonclustered covering index)

Drop the existing indexes using below query and run the stats select queries.

DROP INDEX IX\_Coffee\_Vending\_Machine\_Data\_MachineID\_Date\_Time ON Coffee\_Vending\_Machine\_Data

DROP INDEX IX\_Coffee\_Vending\_Machine\_Data\_ID ON Coffee\_Vending\_Machine\_Data

TRUNCATE TABLE log\_simulation\_job\_execution

[](https://brahmanandtech.files.wordpress.com/2020/04/image-39.png)

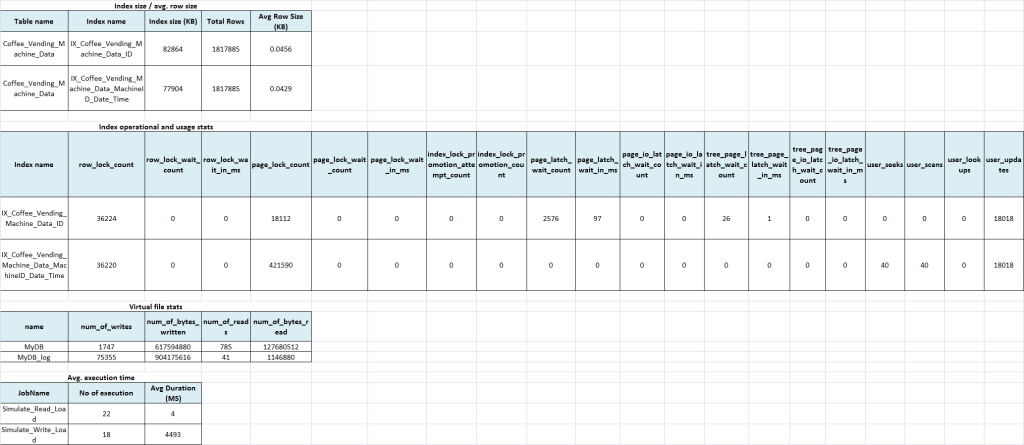
Create the below indexes, enable all the jobs and disable it after 2 minutes. Once all the jobs are disabled run the stats select queries.

CREATE CLUSTERED INDEX IX\_Coffee\_Vending\_Machine\_Data\_ID ON Coffee\_Vending\_Machine\_Data (ID)

CREATE NONCLUSTERED INDEX IX\_Coffee\_Vending\_Machine\_Data\_MachineID\_Date\_Time

ON Coffee\_Vending\_Machine\_Data (MachineID, Date\_Time)

INCLUDE (Sensor1\_Value, Sensor2\_Value, Sensor3\_Value, Sensor4\_Value)

[](https://brahmanandtech.files.wordpress.com/2020/04/image-40.png)

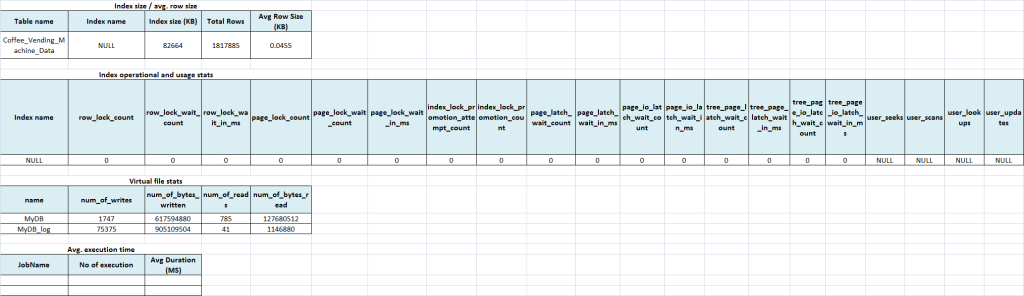
Execution#4 with Natural Clustered Key

Drop the existing indexes using below query and run the stats select queries.

DROP INDEX IX\_Coffee\_Vending\_Machine\_Data\_MachineID\_Date\_Time ON Coffee\_Vending\_Machine\_Data

DROP INDEX IX\_Coffee\_Vending\_Machine\_Data\_ID ON Coffee\_Vending\_Machine\_Data

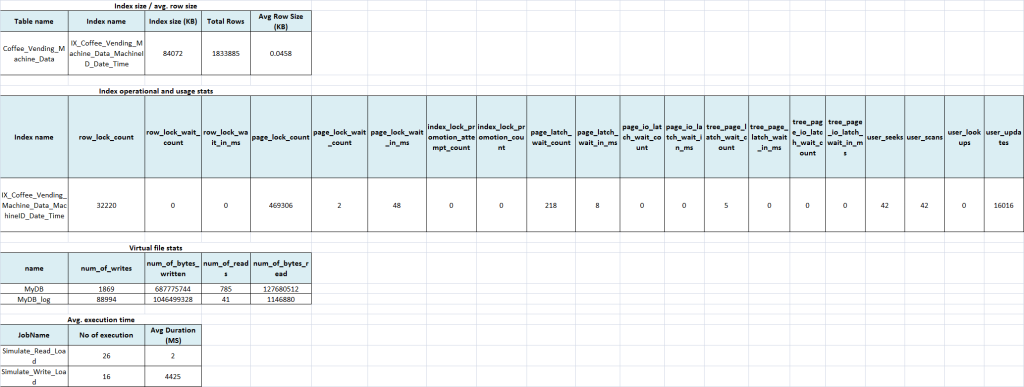
TRUNCATE TABLE log\_simulation\_job\_execution

[](https://brahmanandtech.files.wordpress.com/2020/04/image-41.png)

Create the below indexes, enable all the jobs and disable it after 2 minutes. Once all the jobs are disabled run the stats select queries.

CREATE CLUSTERED INDEX IX\_Coffee\_Vending\_Machine\_Data\_MachineID\_Date\_Time

ON Coffee\_Vending\_Machine\_Data (MachineID, Date\_Time)

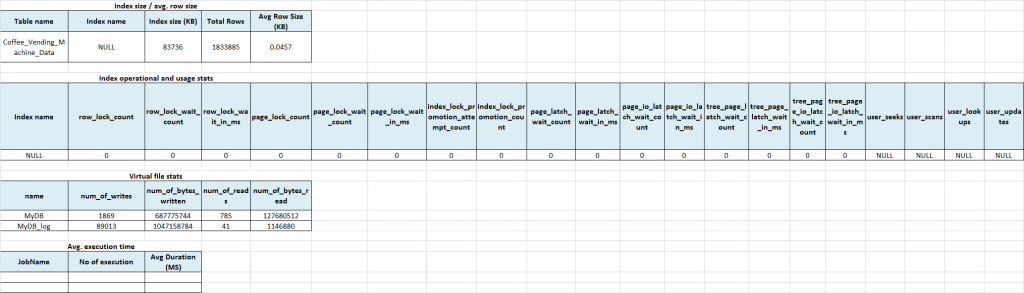
[](https://brahmanandtech.files.wordpress.com/2020/04/image-42.png)

Execution#5 without any Index

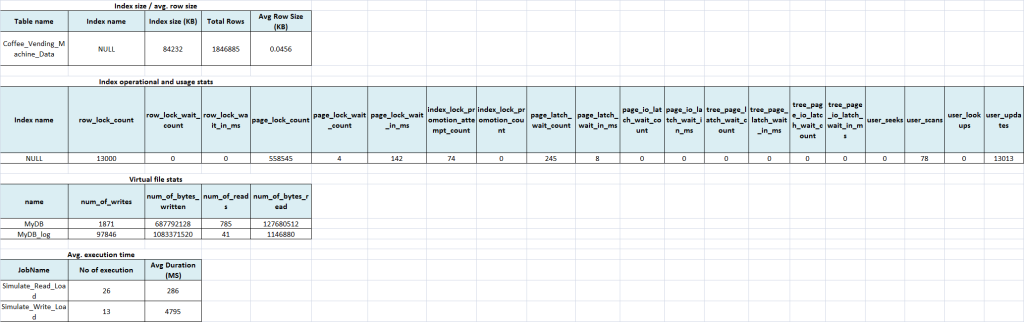
Drop the existing index using below query and run the stats select queries.

DROP INDEX IX\_Coffee\_Vending\_Machine\_Data\_MachineID\_Date\_Time ON Coffee\_Vending\_Machine\_Data

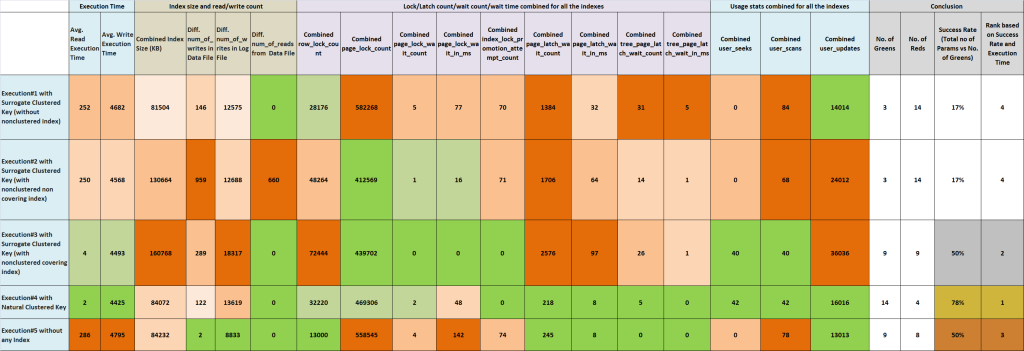
TRUNCATE TABLE log\_simulation\_job\_execution

[](https://brahmanandtech.files.wordpress.com/2020/04/image-43.png)

Enable all the jobs and disable it after 2 minutes. Once all the jobs are disabled run the stats select queries.

[](https://brahmanandtech.files.wordpress.com/2020/04/image-44.png)

Findings

[](https://brahmanandtech.files.wordpress.com/2020/04/image-47.png)

* **Execution#4 with Natural Clustered Key** won with **GOLD**. It stood 1st in the Ranking in our experiment.
* **Execution#3 with Surrogate Clustered Key (with nonclustered covering index)** was runner up with **SILVER**. It stood 2nd in the Ranking in our experiment. It took the additional storage space which in-turn is an overhead to Disk IO.
* **Execution#5 without any Index** stood **BRONZE**. It stood 3rd in the Ranking in our experiment.
* **Execution#1** and **Execution#2** both lost the game in our experiment. Both of them stood 4th in the Ranking in our experiment.

Conclusion

There is no panacea solution. It’s always advisable to evaluate every recommendation / best practice and its trade-offs holistically, before actually applying it.

Sometimes it’s wise decision not to have any index (including clustered index), than having additional overhead of unused / poorly performing index.